Software acceleration of CARP, an iterative linear solver and preconditioner

Art Petrenko, Tristan van Leeuwen and Felix J. Herrmann, Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia*

SUMMARY

We present the results of software optimization of a row-wise preconditioner (Component Averaged Row Projections) for the method of conjugate gradients, which is used to solve the diagonally banded Helmholtz system representing frequency domain, isotropic acoustic seismic wave simulation. We demonstrate that in our application, a preconditioner bound to one processor core and accessing memory contiguously reduces execution time by 7% for matrices having on the order of 10^8 non-zeros. For reference we note that our C implementation is over 80 times faster than the corresponding code written for a high-level numerical analysis language.

INTRODUCTION

Simulation of seismic wave propagation is the biggest computational burden of full-waveform inversion (FWI). In the frequency domain, and specializing to the constant density isotropic acoustic case for concreteness, this simulation corresponds to solving a large linear system of the form

$$\left(\omega^2\mathbf{m}+\frac{1}{\xi_x(x)}\frac{\partial}{\partial x}\frac{1}{\xi_x(x)}\frac{\partial}{\partial x}+\frac{1}{\xi_y(y)}\frac{\partial}{\partial y}\frac{1}{\xi_y(y)}\frac{\partial}{\partial y}+\frac{1}{\xi_z(z)}\frac{\partial}{\partial z}\frac{1}{\xi_z(z)}\frac{\partial}{\partial z}\right)\mathbf{u}=\mathbf{q}, \quad (1)$$

where $\mathbf{m}$ is a vector of medium properties (for example, a slowness squared model), $\mathbf{u}$ is the (complex) Fourier transform of the pressure with respect to time, $\mathbf{q}$ is the amplitude of the source at angular frequency ω, and the ξ's are perfectly matched layer functions that eliminate reflection artifacts from the boundaries of the domain. The equation can be succinctly written in matrix notation, $H\mathbf{u}=\mathbf{q}$, where H is the $N\times N$, Helmholtz operator. The matrix H is sparse and has diagonal band structure. The vectors $\mathbf{m}$, $\mathbf{q}$ and $\mathbf{u}$ have one element per physical domain grid point. At the beginning of the inversion process, an initial guess for $\mathbf{m}$ is provided, and at each FWI iteration (1) is solved once for each source and frequency, the resulting wavefields $\mathbf{u}$ are compared against measured data, and based on the difference an updated $\mathbf{m}$ is calculated. The process repeats until the required accuracy for medium properties has been reached. Since the matrix H is very large (target size $\approx 10^{10}$ non-zeros) direct solvers are unsuitable for this procedure because they destroy the sparse structure of the matrix by creating infill, and use too much memory. Instead we use an iterative method: the method of conjugate gradients with the CARP preconditioner known as CARP-CG, developed by Gordon and Gordon (2005, 2010) and in turn based on the algorithm for solving linear systems due to Kaczmarz (1993), originally published in 1937.

The Kaczmarz algorithm solves a general linear system $A\mathbf{x}=\mathbf{b}$ by projecting the current iterate $\mathbf{x}_i$ onto the hyperplane orthogonal to a row of the matrix, and setting the next iterate equal to the result:

$$\mathbf{x}_{i+1}=\mathbf{x}_i+w(b[k]-\langle\mathbf{a}_k,\mathbf{x}_i\rangle)\frac{\mathbf{a}_k^*}{\|\mathbf{a}_k\|^2}. \quad (2)$$

Here w is a relaxation parameter, $\mathbf{a}_k$ is the k^{th} row of A, $b[k]$ is the corresponding element of the right-hand side, and $*$ denotes Hermitian transpose. A Kaczmarz "sweep" consists of i and k jointly running from the first row to the last (forward), from the last to the first (backward), or from first to last and back again (double). CARP builds on the Kaczmarz algorithm by performing double sweeps on sets of rows in parallel, averaging those elements of $\mathbf{x}_i$ that are updated by sweeps on more than one set. If CARP (or Kaczmarz) is run repeatedly, $\mathbf{x}_i$ will converge to the solution of $A\mathbf{x}=\mathbf{b}$, but instead we use the algorithm as a preconditioner for CG. First we represent the double CARP sweep operator in matrix notation and apply it to the Helmholtz system:

$$\text{DCSWP}(H,\mathbf{u},\mathbf{q},w)=\mathbf{u}=Q\mathbf{u}+R\mathbf{q}, \quad (3)$$

noting that since $\mathbf{u}$ is the solution, it is an eigenvector of DCSWP. Q summarizes the action of the first and last terms in (2) over the $2N$ iterations of a double sweep, and R the action of the middle term. This leads to an equivalent formulation of the problem:

$$(I-Q)\mathbf{u}=R\mathbf{q}. \quad (4)$$

A solution $\mathbf{u}$ to (4) will also solve (1). Furthermore, because the matrix Q is a product of symmetric matrices Q_i each corresponding to a single CARP iteration, performed in the symmetric order of a double sweep, $I-Q$ itself is symmetric and positive semi-definite (unlike H). Thus the system is amenable to solution with CG, as shown by Gordon and Gordon (2010) and further demonstrated by van Leeuwen et al. (2012).

CARP-CG is presented as Algorithm 1, where because we are solving the preconditioned system, the matrix vector product (shown on line 6) is with $I-Q$ and is implemented using a double CARP sweep with the right hand side set to $\mathbf{0}$. References to the "CARP iterate" indicate intermediate vectors in the calculation of $\mathbf{s}_i$, in contrast to the CG iterate, which is $\mathbf{u}_i$. Since the double CARP sweep implements the only matrix-vector operation in CARP-CG, it is the most time-consuming

Algorithm 1 CARP-CG, Gordon and Gordon (2010)

```
Input: $H$, $\mathbf{u}_0$, $\mathbf{q}$, $w$
 1: $i = 0$
 2: $R\mathbf{q} = \mathrm{DCSWP}(H, \mathbf{0}, \mathbf{q}, w)$
 3: $\mathbf{r}_0 = R\mathbf{q} - \mathbf{u}_0 + \mathrm{DCSWP}(H, \mathbf{u}_0, \mathbf{0}, w)$
 4: $\mathbf{p}_0 = \mathbf{r}_0$
 5: while $\|\mathbf{r}_i\|^2 > tol$ do
 6:     $\mathbf{s}_i = (I - Q)\mathbf{p}_i = \mathbf{p}_i - \mathrm{DCSWP}(H, \mathbf{p}_i, \mathbf{0}, w)$
 7:     $\alpha = \|\mathbf{r}_i\|^2 / \langle \mathbf{p}_i, \mathbf{s}_i \rangle$
 8:     $\mathbf{u}_{i+1} = \mathbf{u}_i + \alpha \mathbf{p}_i$
 9:     $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha \mathbf{q}_i$
10:     $\beta = \|\mathbf{r}_{i+1}\|^2 / \|\mathbf{r}_i\|^2$
11:     $\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \beta \mathbf{p}_i$
12:     $i = i + 1$
13: end while
Output: $\mathbf{u}$
```

part of that algorithm, and we devoted efforts to optimize its implementation. Related work has been done by Elble et al. (2010), who have examined the improvements that Kaczmarz and other algorithms experience on Graphical Processing Units (GPUs).

METHODS

We constructed the wave equation operator H using a finite-difference stencil designed to minimize numerical dispersion, as in Operto et al. (2007), and normalized its rows. When working in a three dimensional physical domain each row has up to 27 non-zero elements corresponding to the neighbors of a given point; thus H has 27 non-zero diagonals. These diagonals were stored in a new array, and the diagonal offsets in a short vector *idx*. Since the main operation in CARP is taking dot products with rows of H, the diagonals were stored with the rows contiguous in memory, allowing for efficient access. While we did implement a multi-threaded version of CARP, degree of parallelism is not directly relevant to the results we present and the performance of that implementation has not yet been adequately measured. Hence below we refer only to single-threaded Kaczmarz sweeps.

To study performance of the Kaczmarz sweeps we programmed them in C, reading H (in the format described above), $\mathbf{u}_0$ and $\mathbf{q}$ from files stored on disk. Complex numbers were stored as arrays of structures, and vector calculations were programmed directly with `for` loops. The program was compiled with the `-O2` optimization switch, as well as the `-xavx` switch, to take advantage of the vector processing capabilities of the CPU. To ensure a predictable profiling environment, the simulations were bound to one hardware thread on one core of an Intel Xeon E5-2670 machine.

The medium $\mathbf{m}$ was taken to be the SEG/EAGE overthrust velocity model, presented by Aminzadeh et al. (1997), undersampled by various factors. The initial

```
    for(i=start_row;i<end_row;i++){
146   c.real = b[i+(long)idx[ncol/2]].real;
      c.imag = b[i+(long)idx[ncol/2]].imag;
148   for(j=0;j<ncol;j++){
        k = i + (long)idx[j] - (start_row-
            firstdims);
150     if(0<=k && k<ny){
          c.real -= R[i*ncol + j].real*y[k].
              real - R[i*ncol + j].imag*y[k].
              imag;
152       c.imag -= R[i*ncol + j].real*y[k].
              imag + R[i*ncol + j].imag*y[k].
              real; } }
      c.real *= w;
154   c.imag *= w;
      for(j=0;j<ncol;j++){
156     k = i + (long)idx[j] - (start_row-
            firstdims);
        if(0<=k && k<ny){
158       y[k].real += c.real*R[i*ncol + j].
              real + c.imag*R[i*ncol + j].imag;
          y[k].imag += c.imag*R[i*ncol + j].
              real - c.real*R[i*ncol + j].imag;
160 } } }
```

Listing 1: C code for forward Kaczmarz sweeps. Here, `R` is the array of diagonal bands of H, `y` is the Kaczmarz iterate ($\mathbf{p}_i$ in the case of CARP-CG), `b` represents the right hand side, and `c` is a variable that holds intermediate results in the calculation of $w(b[k] - \langle \mathbf{a}_k, \mathbf{x}_i \rangle)$.

guess was a vector of normally distributed random elements, and the source $\mathbf{q}$ was also i.i.d. Gaussian, but multiplied element-wise by the same vector of row norms used to normalize H.

The computationally intensive part of the (forward) Kaczmarz sweeps is shown in Listing 1. Each iteration of the outside loop over `i` corresponds to one Kaczmarz step. Inside this outer loop are two loops over the non-zero diagonal bands of H, running over `j`. The first loop (lines 148–152) calculates the component of the Kaczmarz iterate along row $\mathbf{a}_i$, and the second loop (lines 155–160) adds a multiple of $\mathbf{a}_i$ to the iterate.

In the process of accelerating the Kaczmarz sweep implementation, we followed the methodology outlined by Intel (2012). First we identified blocks of code where the program was spending the most time. Then we measured efficiency of those blocks by recording the fraction of time that the processor pipeline spent dealing with each of four categories of operations: retired, canceled, back-end bound and front-end bound. Retired operations are those that have successfully finished their calculation, canceled operations are due to mispredicted execution branches, back-end bound operations are waiting on either computational or memory access resources, and front-end operations are stalled because of not being able to timely deliver the next instruction to the pipeline. Ideally, most operations would be retired, with few appearing in the other cate-

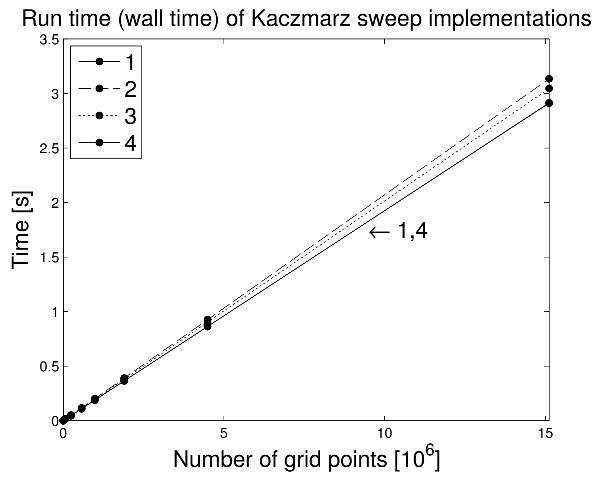

Figure 1: Average time over 10 sweeps; note that the curves for implementations 1 and 4 overlap. For the last set of points, which correspond to a $94 \times 401 \times 401$ physical domain, the speed-up between implementations 4 and 2 is 7%. For comparison, the speed-up we measured between sweeps programmed in MATLAB (not shown) and any of the C-sweeps is approximately 80-fold.

gories. Once the general nature of an inefficiency within an often-used block was established, we would examine both the source code and the resulting assembly code for ways to improve our design.

RESULTS

In CARP-CG the forward and backward sweeps are run one after the other, but during testing we noticed that the forward sweeps were consistently faster. Our hypothesis was that this was due to different patterns of memory access. While the forward sweeps accessed matrix elements in the order in which they were stored (both indices `i` and `j` in Listing 1 increasing), backward sweeps jumped from the end of one row to the start of the previous row (`i` decreasing and `j` increasing).

Implementation	Sweep	`j`-index
1	forward	increment
2	backward	increment
3	forward	decrement
4	backward	decrement

Table 1: Definition of the parameters that are varied in order to accelerate the Kaczmarz sweeps.

To test the hypothesis, we ran the sweeps with four combinations of parameters shown in Table 1. The execution time (wall time) is shown in Figure 1. As expected, those combinations that correspond to accessing memory sequentially ran faster that those that had to make jumps. However while there is a clear correlation between execution time and the fraction of the pipeline containing retired operations per cycle, the correlation with back-end bound operations (the ones that would be waiting on memory access) is more muddled. As can be seen from Figure 2, the front-end bound fraction also changes across implementations. Thus while we observe that incrementing `j` in a forward sweep and decrementing it in a backward sweep decreases execution time, we cannot yet definitively conclude that our hypothesis of contiguous memory access is the correct explanation for the improvement in timing seen from implementation 3 to 1 and from 2 to 4.

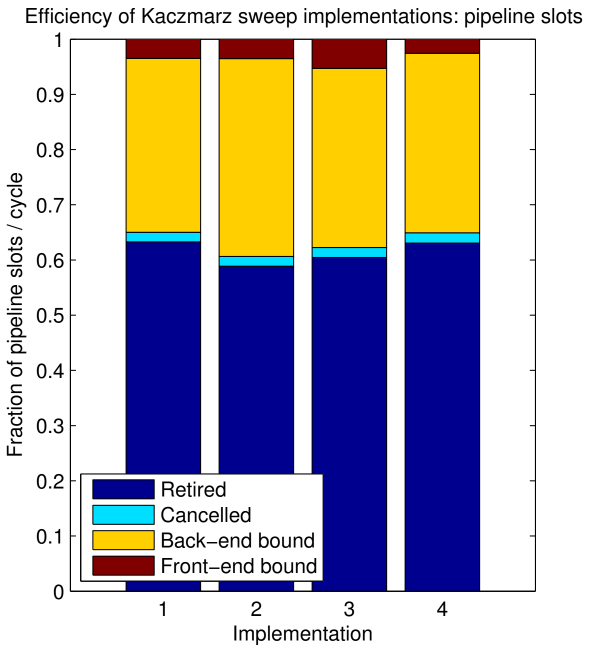

Figure 2: Average performance over 10 runs using a $94 \times 401 \times 401 \approx 15 \cdot 10^6$grid points subsampling of the SEG/EAGE overthrust model. A pipeline "slot" is one pipeline unit in the processor.

CONCLUSIONS

The results presented above are a demonstration of the well known rule that memory is best addressed in order. It is also worth noting that improvements in efficiency metrics such as the percentage of time the pipeline is waiting for memory access, do not translate directly into improvements in execution time.

We are currently undertaking further work with the CARP preconditioner. This includes measuring the effect of replacing the direct `for` loops in the body of the sweeps with calls to CBLAS, an optimized linear algebra library, and expanding the scope of the implementation to include the whole CG method (parallelized via multi-threading). We are also porting the algorithm onto FPGAs, and look forward to being able to report on these developments in the future.

ACKNOWLEDGEMENTS

We would like to thank Henryk Modzelewski for his support of the infrastructure necessary to carry out this research. This work was in part financially supported by the Natural Sciences and Engineering Research Council of Canada Discovery Grant (22R81254) and the Collaborative Research and Development Grant DNOISE II (375142-08). This research was carried out as part of the SINBAD II project with support from the following organizations: BG Group, BGP, BP, Chevron, ConocoPhillips, Petrobras, PGS, Total SA, and WesternGeco.

REFERENCES

Aminzadeh, F., B. Jean, and T. Kunz, 1997, 3-D salt and overthrust models: Society of Exploration Geophysicists.

Elble, J. M., N. V. Sahinidis, and P. Vouzis, 2010, GPU computing with Kaczmarz's and other iterative algorithms for linear systems: Parallel Computing, **36**, 215–231.

Gordon, D., and R. Gordon, 2005, Component-averaged row projections: A robust, block-parallel scheme for sparse linear systems: SIAM Journal on Scientific Computing, **27**, 1092–1117.

———, 2010, CARP-CG: A robust and efficient parallel solver for linear systems, applied to strongly convection dominated PDEs: Parallel Computing, **36**, 495–515.

Intel 2012, Intel 64 and IA-32 architectures optimization reference manual: http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf. (Order Number: 248966-026).

Kaczmarz, S., 1993, Approximate solution of systems of linear equations: International Journal of Control, **57**, 1269–1271.

Operto, S., J. Virieux, P. Amestoy, J.-Y. L'Excellent, L. Giraud, and H. B. H. Ali, 2007, 3D finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study: Geophysics, **72**, SM195–SM211.

van Leeuwen, T., D. Gordon, R. Gordon, and F. J. Herrmann, 2012, Preconditioning the helmholtz equation via row-projections: Presented at the EAGE technical program, EAGE.